Vircon32:Cコンパイラーのためのガイド

資料作成日2023.03.11年Carra著

これは何だ?

このドキュメントは、C言語を使用してVircon32コンソールでプログラムを作成する方法についてのクイックガイドです。このガイドの目的は、この言語の機能と、標準および他のコンパイラと比較した場合の違いを示すことです。このドキュメントは、コンパイラバージョン23.3.7に適用されます。

まとめ

このドキュメントは、Vircon32 C言語とそのコンパイラの機能の1つをカバーするセクションで構成されています。

[Summary 1](#_Toc125281794)

[Introduction 2](#_Toc125281795)

[Comments 2](#_Toc125281796)

[Data types 3](#_Toc125281797)

[Literal values 5](#_Toc125281798)

[Variables 6](#_Toc125281799)

[Functions 8](#_Toc125281800)

[Expressions 9](#_Toc125281801)

[Flow control 10](#_Toc125281802)

[Embedding assembly code 13](#_Toc125281803)

[The preprocessor 13](#_Toc125281804)

[The standard library 15](#_Toc125281805)

[Global differences with standard C 16](#_Toc125281806)

[General compiler limitations 17](#_Toc125281807)

[Using the compiler 18](#_Toc125281808)

はじめに

CコンパイラはVircon32開発ツールの一部であり、その機能は、C言語で記述されたプログラムを解釈してアセンブリ言語に変換することです。次に、2番目のプログラム(アセンブラ)がそれをVircon32 CPUが実行できるマシンコード命令に変換します。



このコンパイラで使用されるC言語は標準とほとんど同じですが、プログラミング時に考慮する必要のあるいくつかの違いと制限があります。これらの違いのいくつかは、コンソール自体の動作方法によって与えられますが、他は単にコンパイラをプログラミングするときの私自身の制限の結果です。それでも、アセンブラを学ぶ必要があることによってコンソール上でゲームを作成するための障壁を設けるよりも、たとえいくつかの制限があっても、より高いレベルでプログラムできることの方がはるかに価値があると思います。

次のセクションでは、この言語の特徴について説明します。赤色で強調表示する標準のCコンパイラとの違い。これにより、すでにC言語でのプログラミング方法を知っている人は、注意して考慮すべき主な事項を一目で確認できます。

コメント

コメントは、コンパイラが解釈しようとすることなく、独自の説明をプログラムに含める方法です(これによりエラーが発生します)。つまり、コメントはプログラム自体の一部ではありません。コメントには2種類あり、標準C言語と同じように機能します。

## 行コメント

これらのコメントは次で始まります。//////で終了し、行の最後で終了します。行が次の行で終了することによって継続される場合は、次の行まで延長できます。\です。

|  |
| --- |
| //継続する行コメント\  次の行に  内部x=7+5//コメントは続きません  内部y=xである。 |

## ブロックコメント

ブロックコメントは次で始まります。/\*(該当日本語なしで終わるまで終わらない\*/\*/\*/(別の行で実行された場合でも)。

|  |
| --- |
| /\*このコメントは、次の行を否定します  int x=7+5;  \*/\*/\*/  内部y; |

データ型

言語によって処理されるすべての値または変数は、その値の解釈方法を示すデータ型に属します。ここでは、使用できるデータ型を確認します。

## 基本タイプ

Vircon32 C言語は、標準C言語とは異なりには、サイズが32ビットのデータタイプしかありません。さらに、これらのタイプのバリアントはありません。符号(signed、unsigned)にも、サイズ(short、long)にもありません。

Vircon32は、メモリを32ビットワードで構成します。このコンソールでは、メモリの最小単位がワード(バイトではなく)であるため、次のことを考慮する必要があります。

* データ型のすべてのサイズはバイト単位ではなくワード単位で測定される
* すべてのメモリアドレスとオフセットはバイト単位ではなく、ワード単位で指定されます。

このコンパイラには、次の4つの基本データ型があります。

|  |
| --- |
| int float bool void(int浮動小数点ブール値 |

これらの型はすべてサイズ1(つまり32ビット)を持ちます。void型は特殊で、関数が値を返さないことを示すためにのみ使用できます。

## 派生タイプ

配列とポインタの基本的な型を使用して新しい型を作成できます。これは、次のように複数回実行できます。

|  |
| --- |
| 内部※印空所\*\*\*\*\*\*浮く[3][5]空所\*[4] |

voidを使用してポインタを作成できますが、配列は作成できません(void型の値が存在しないため)。voidへのポインタは、情報を解釈せずに読み取りまたは保存するメモリアドレスを格納するために使用されます(voidは型が存在しないことを示すために使用されます)。

## 関数ポインタ

現在のコンパイラバージョンには関数ポインタはありませんが、次のバージョンで追加される機能として計画されています。

## 複合型

構造と共用体を作成することによって、いくつかのdayaタイプをより大きなタイプにグループ化できます。これらのグループの各メンバーは、名前でアクセスされるフィールドです。

標準Cでは、struct/enumを宣言すると型ではなく変数が宣言されます(typedefが使用されていない場合)。Vircon32で使用されているC言語は、C++がクラス宣言を扱うのと似た方法で、これらを型宣言として扱います。

|  |
| --- |
| //Point型を宣言  構造体[点]  {を  内部x,y;x,y;x  }  //Word型を宣言  ユニオンワード  {を  内部AsInteger  空所※印AsPointer  }  //いくつかの構造体を使用  点P1、P2  P1.x=10  P1.y=-7  P2=P1  //いくつかの共用体を使用  単語W;  W.AsInteger=0xFF110AF  int\*ポインタ=W。AsPointer |

構造体や共用体には、他の構造体や共用体を含めることもできます。構造体や共用体自体を含めることもできますが、明らかな理由からポインタを使用する必要があります。

標準Cは、構造体が個々のビットをグループ化するためにビットフィールドを使用することを可能にする。これらはここではサポートされておらず、追加する予定もありません。

## 列挙

一連の整数定数を定義し、それらを独自の型にグループ化することができます。C言語では、列挙によってこれを可能にします。一般に、これらの値は整数として扱われ、同じ方法で使用できますが、型はより限定的です。structおよびunionと同様に、enumを使用すると、typedefを使用した場合と同様に常に型が宣言されます。

値が指定されていない場合、最初の定数の数値は0になり、後続のすべての定数の値は前の値+1になります。

|  |
| --- |
| //タイプSemaphoreを宣言する  列挙値セマフォー  {を  赤=1  黄  緑  }  //これらの操作は正常です。  セマフォーS1、S2  S1=赤色  S2=S1である。  内部値=黄+緑//安全にenumをintに変換できる  //これらの割り当てはエラーを生成します!  S1=<数値>緑-赤  S2=1//Redと同じ値ですが、intをenumに変換できません |

## typedefによる型の定義

使用できるのはタイプ定義データ型に名前を付けます。その名前を使用して、そのデータ型を表すことができます。

|  |
| --- |
| //タイプを定義する  typedef int(型定義整数)\*\*int\_ptr2;<整数>  //これを使用して  int\_ptr2 P=<文字列>特殊 |

次の点に注意してください。このコンパイラーでは、標準Cに比べて複合型の作成方法が単純化されています。たとえば、通常のCでは、配列へのポインタを定義するには、括弧を使用して名前を型自体の内部に配置する必要があります。(該当日本語なし

|  |
| --- |
| typedef int(型定義整数)(\*ptr\_to\_array)[5];<配列> |

代わりに、このコンパイラでは、これらは常に<type><name>のように別々に保持されます。これにより、右から左に読み取ることで、タイプを識別しやすくなります。

|  |
| --- |
| typedef int(型定義整数)[5]\*ptr\_to\_array(配列へのptr); |

リテラル値

私たちのプログラムでは、文字どおりに記述された定数値を使用できます。これらの値のそれぞれが表すデータ型に応じて、異なる表記と数値表現があります。このコンパイラには次のものがあります:

|  |
| --- |
| -15インチ//10進数のint  0xFF1A、0 xFF 2 A、0 x//int(16進数)  0.514人//浮動小数点  真//ブール  「a」は//intを文字として使用(charは存在しません)  "ホラ"//int[5](4文字の文字列+終了0)  特殊//NULLポインタ |

ブールリテラルの値は、true=1、false=0です。

ほとんどのCコンパイラではNULLの値は0ですがこのコンパイラでは、値は実際には-1です。Vircon32では、メモリアドレス0が有効なアドレスであるため、このように選択されました。

その場合でも、NULLを使用した操作は通常どおりに実行できます。たとえば、次のようなチェックを行います。if(ポインタ)はポインタが-1と異なるかどうかをテストする。

浮動小数点値の場合、科学表記(例:1.35e-7)は現在サポートされていません。

変数

定数データに加えて、変数も処理できます。変数は、変化する値が格納されるメモリアドレスです。この値にアクセスするには、変数を識別する名前と、格納された値を解釈するデータ型を使用します。

## 変数の宣言

変数は型と名前で宣言され、オプションで値で初期化できます。

|  |
| --- |
| 浮く[速度]=2.5  ブール有効: |

typedefについて話したときに見たようにこのコンパイラの型は、常に名前とは別に保持されます。この場合は、<type><name>(=<value>)です。たとえば、このコンパイラで配列を宣言するには、次のようにする必要があります。

|  |
| --- |
| //これは、それぞれが10個のintを持つ20個の配列を持つ配列です。  内部[20][10](英語)LevelBricks |

標準Cのように配列を宣言しようとすると、コンパイラでエラーが発生します。

## 複数の宣言

1つの宣言で同じ型の複数の変数をカンマで区切って宣言することもできます。

|  |
| --- |
| //intへの3つのポインタを宣言する  内部\*ptr1=&数値,ptr2=ptr1,ptr3; |

次の点に注意してください。このコンパイラでは、同時に宣言されたすべての変数は常に同じ型です(ここでは、すべてint\*です)。標準Cでは最初のものだけがポインタになります。

## 初期化リスト

配列および構造体は、複数の値のリストを使用して初期化できます。これらのリストは、次の例に示すように、より複雑なタイプに対してネストすることもできます:

|  |
| --- |
| //構造体を初期化する  構造体[点]  {を  内部x,y;x,y;x  }  点P={3,-7}  //構造の配列にネストされたリストを使用する  点[3]TrianglePoints={{0,0},{1,0},{1,1}}  //int配列の場合、リストの代わりに文字列を使用することもできます。  内部[10]を参照してください。テキスト=「こんにちは」//慎重に、C終端として1文字(0)を追加 |

## 変数の種類

C言語には、ローカル変数とグローバル変数という2種類の基本的な変数があります。

* ローカル変数は関数の本体で宣言され、そのスコープ内でのみアクセス可能です(スタックに格納され、実行によって変更されるため)。
* グローバル変数は関数の外部で宣言され、アドレスが固定されているため、宣言後にプログラム全体からアクセスできます。

Vircon32 Cでは、埋め込みファイルという3つ目のタイプの変数が追加されました。埋め込み次に例を示します。

|  |
| --- |
| //この変数の内容は、コンパイル時にファイルから読み込まれます。  埋め込みint[200][100]<数値>TileMap="GameData\\LevelMap.bin" |

この場合、コンパイラはその配列をRAMメモリではなくカートリッジに格納します。これにより、カートリッジ内のデータは常に読み取り専用ですが、コードに書き込むことなく大量のデータを使用できます。このメカニズムにより、コンピュータで行われるように外部にロードするファイルがない場合でも、プログラムで外部データを使用できます。

コンパイラでは、既存のファイルに加えて、このファイルがそれを格納する変数と同じサイズであることが必要であることを知っておくことが重要です。つまり、この場合、ファイルLevelMap.binファイルのサイズは次のようになります:

整数の数=200 x 100=20,000

バイト/int=4(これらは32ビットintです)

合計サイズ=80,000バイト

埋め込みファイルはグローバルスコープでのみ使用できます。つまり、関数内には表示できません(スタックに格納されないため)。

## モディファイヤ

このコンパイラでは、現在のところ、変数宣言に適用する修飾子はありません。const、static、volatile、registerはサポートされていません。将来実装される可能性があるのは、constだけです。

関数

C言語では、宣言だけでなく、関数の外部で任意のステートメントを実行できるようになりました。プログラム内のコードを実行するには、実行するステートメントを含む関数を宣言する必要があります。Vircon32 Cの関数は、標準C(ただし、すでに説明したように、このコンパイラーのデータ型表記を使用します。)をクリックします。例:

|  |
| --- |
| //2つの整数を加算する関数  内部AddValues(内部a内部ロ)  {を  返却a+b  }をクリックし |

関数の本体には、順次実行される複数のステートメントを含めることができます。

標準Cとの重要な違いは、コンパイラの制限により関数は、1以外のサイズのパラメータや戻り値を使用することはできません。つまり、配列、共用体、または構造体を使用することはできません(サイズが1つの単語でない場合)。代わりに、それらはそれらへのポインタで操作する必要があります。

しかし、標準C言語と同じように、配列がポインタに分解されるときに、パラメータとして「配列を渡す」ことができます。つまり、次のようにします。

|  |
| --- |
| //N個の整数を加算する関数  内部合計値(int\*(整数)値内部数)  {を//.//.//}をクリックし  //10個の整数の配列  内部[10]値  //配列の最初の5つの値を加算する  内部Sum=SumValues(値,5); |

C言語では、関数の引数リストに省略記号を使用して、不定数のパラメータを受け取ることができます。これは、printfなどの標準関数で使用されます。このメカニズムは、このコンパイラではサポートされていません。

式

式は、変数とリテラル値を組み合わせることによって実行できる操作です。これらの組み合わせは、結果として単一の最終値が得られるまで、関数または演算子を適用することによって作成できます。

## 関数呼び出し

括弧とカンマの標準表記を使用して、すでに宣言されている任意の関数を呼び出すことができます。

|  |
| --- |
| //変数と定数を追加  内部Sum=SumValues(変数,5);  //関数呼び出しをネストできる  PrintNumber(SumValues(変数,5)); |

関数が値を返さない場合(戻り値の型がvoidの場合)、その関数の呼び出しは、結果を生成しない式であるため、他の式では使用できません。

## 演算子

ほとんどの場合、演算子は、関数として記述されるのではなく、オペランド間に記号(+や-など)として挿入される数学的操作を表します。このコンパイラでは、標準C言語と同じ演算子が使用され、優先順位と関連性も同じです。このコンパイラでは、右ビットシフト演算子(>>、>>=)の動作は、多くのCコンパイラで使用されている算術シフトではなく、論理ビットシフトです。

三項演算子aからb:cおよびカンマ演算子ア、イは、このコンパイラでサポートされていない唯一の標準C演算子です。

## 特殊演算子

の関数sizeof()演算子は、データ型または式の結果のサイズ(常に32ビットワード単位)を決定します。

|  |
| --- |
| //式の結果のサイズ  内部サイズ1=サイズ(2+5)//サイズ1=1(int)  //データ型のサイズ  内部サイズ2=サイズ(の内部[2][5])//サイズ2=10 |

次に、C言語は必要に応じてデータ型を暗黙的に変換しますが、式を別のデータ型に明示的に変換する演算子も備えています。

|  |
| --- |
| 浮くx=5.0/秒(の内部)2.5;//2.5は切り捨てられるため、x=5.0/2=2.5 |

フロー制御

Cプログラムは、式のシーケンスを使用して構築されるだけでなく、実行フローを制御するためのツールも必要です。次のものを使用できます。

## 条件

実行を継続する場所を制御する最も基本的な方法は、単純な条件です。もしおよびそれ以外私たちは条件を評価し、それが満たされた場合に何が起こるか、満たされなかった場合に何が起こるかを決定することができます。

また、いくつかのifをチェーンして、関連する条件をチェックすることもできます。

|  |
| --- |
| //チェーニングされたif  もし(x>0)  印刷(「正」)  else if文(x<0の場合)  印刷(「負」)  それ以外  {を  印刷(「ゼロ」)  ShowAlert x  }をクリックし |

## スイッチ

複数の整数値から選択する必要がある場合は、複数のif-else条件をチェーンする代わりにスイッチ一連のケースから選択します。

|  |
| --- |
| スイッチ(WeaponPowerLevel)  {を  //中電力の影響  事件1点:  事件2点:  Player.健康=WeaponPowerLevel;  ブレーク  //プレイヤーを破壊するハードインパクト  事件3点:  MakePlayerExplode x  ブレーク  //影響が弱すぎる場合、またはエラーの可能性がある場合  デフォルト(該当日本語なし  MakeBulletBounce x  ブレイク;  }をクリックし |

## ラベルと移動

関数内にラベルを定義して、コード内の特定の位置にマークを付けることができます。移動プログラムの実行をこれらの位置にジャンプさせます。

|  |
| --- |
| //ラベルは関数内でのみ有効  ボイドProcessData(データ構造\*データ)  {を  ステップ1:  //場合によっては、この手順を実行しない  もし(SkipPreprocessing)  移動ステップ2:    前処理(データ);    ステップ2:  //エラーが十分に低くなるまでデータの処理を継続  もし(!変換(データ))  移動ステップ1:  }をクリックし |

## ループ

ループは、gotoよりも優れた代替手段です。ループを使用すると、条件が満たされるまでプログラムのセクションを繰り返すことができます。Cには3種類のループがあります:一方するおよび対象です。

|  |
| --- |
| //whileは先頭の条件をチェックします。  一方(x!=0)を返します。  x/=2;  //最後に条件をチェックする  する  {を  x-=7;  y=xである。  }をクリックし  一方(x>0)  //forは、より構成可能なループです。  対象(の内部i=-5;i<=5;i+=2)  print\_number(i);<数値> |

## ループ制御

ループの内側ではブレークを使用してループを終了し、終了後も実行を継続します。続けるループの最後に到達することなく、ループの次の繰り返しに進むことができます。

|  |
| --- |
| 一方(敵.EnemyID<MaxEnemyID)  {を  ++敵;    //アクティブな敵のみを処理  もし(!敵アクティブ)  続ける    ProcessEnemy(&プレイヤー,敵);    //プレーヤーが終了した場合、続行する必要はありません  もし(Player.Health<=0)  ブレーク  }をクリックし |

## 関数からの復帰

関数本体の内部では、次のように使用できます。返却関数を終了します。実行は、その関数が呼び出されたプログラム内のポイントに戻ります。

returnは、値を返すためにも使用されます(関数の戻り値の型がvoidでない場合)。その場合、returnは互換性のある型の値とともに使用する必要があります。

|  |
| --- |
| //値を返さない関数  空所DoNothing()関数  {を  返却  }をクリックし  //ポインタを返す関数  int\*(整数)FindLetterA(のint\*(整数)テキスト)を  {を  一方(テキスト)  {を  もし(\*テキスト==「A」キー)を  返却テキスト;    テキスト++;  }をクリックし    返却特殊  }をクリックし |

埋め込みアセンブリコード

Vircon32 C言語では、他のいくつかの実装と同様に、以下を使用してアセンブリ言語命令を組み込むことができます。アセンブリこの機能は、関数本体の内部でのみ使用できます。この構文は、gccコンパイラで使用される構文と非常によく似ており、各命令の記述にはテキスト文字列が使用されます。

|  |
| --- |
| //アセンブリに実装されたC関数  空所サウンドを選択(内部サウンドID)  {を  アセンブリ  {を  //ここでは、パラメータsound\_idの値を使用します。  "mov R0,{サウンドID}"  "出力SPU\_SelectedSound,R0"  }をクリックし  }をクリックし |

また、アセンブラ命令でCプログラムの変数を使用することもできます。そのためには、変数名を中括弧で囲んで記述します。

Cとアセンブラを接続するこの方法にはいくつかの制限があります。名前のみをサポートし、式はサポートしません。たとえば、配列のメンバを読み取るために使用することはできません。また、各命令で参照できる変数は1つだけです。Vircon32 CPUの動作方法のため、1つの命令で複数の変数を使用することはできません。

プリプロセッサ

C言語は、Cプログラム内のすべてのテキストの前処理を実行します。プリプロセッサは、プログラムを構成するファイルを1行ずつ移動してディレクティブを検索し、検索および認識可能なディレクティブを適用します。

ディレクティブはハッシュ文字で始まります番号これは、行の最初の文字である必要があります(空白を除く)。使用できるディレクティブは次のとおりです。

## ディレクティブ#include

使用方法#含むでは、コードのある時点で別のファイルの内容を挿入できます。これは、異なる機能を別々のファイルに分離することによってプログラムを編成できるので便利です。

|  |
| --- |
| //ヘッダーを含める  #include"敵\\ボス.h" |

プリプロセッサは、まず標準ライブラリフォルダでファイルを検索し、次にソースファイル自体のディレクトリで検索します。

Cでは、ファイルパスを引用符の間("path")または角度の間(<path>)に書くことができます。このコンパイラでは引用符しか使用できません。また、パスは他のテキスト文字列と同様に扱われます。「\」などの特殊文字が含まれている場合は、エスケープシーケンスを使用して記述する必要があります。

## ディレクティブ#defineおよび#undef

プリプロセッサは内部変数のリストを保持します(Cプログラム変数と混同しないようにしてください)。#定義の順にクリックし#定義解除定義には値を指定できますが、空にすることもできます。

|  |
| --- |
| //リテラル値を持つ単純な定義  #define BallDiameter 16  //式で前の定義を使用する定義  #defineボール半径(BallDiameter/2)  //他の古い定義を削除する  #undef BallSize |

このコンパイラでは、defineディレクティブにいくつかの制限があります。標準のCプリプロセッサのようにパラメータと一緒に使用することはできません。番号および######文字列を形成するための定義内の演算子はサポートされていません。

## 条件付きディレクティブ

Cでは、プリプロセッサを使用して、コンパイルする必要のあるプログラムの部分と、コンパイルする必要のないプログラムの部分を区切ることができます。このような条件付きコンパイルは、プリプロセッサ変数が定義されているかどうか、およびその値をチェックするディレクティブを使用することによって実現されます。これを可能にする条件付きディレクティブは次のとおりです。#ifdef(英語の可能性あり#ifn定義#その他および#endif(終了)です。

|  |
| --- |
| #define DEBUG[デバッグ]//commmentingこの行によってプログラムが変更される  #ifdefデバッグ  デバッグ=真  #その他  デバッグ=偽  #endif(終了) |

注意すべき1つの制限は、このコンパイラではコンパイルされていない部分は解析されませんが、(文字列や数値などを認識するために)基本的なレベルで読み込まれます。したがって、コンパイルされていない領域でもエラーが発生する可能性があります。

ディレクティブ#もしおよび#elif(英語の可能性あり標準C言語からは現在サポートされていません。

## ディレクティブ#errorおよび#warning

これらのディレクティブを使用すると、コンパイル中にコンパイラにメッセージを表示させることができます。#警告の場合、コンパイラは警告を表示するだけです。一方#エラーコンパイルは中止されます。

|  |
| --- |
| //この行の後もコンパイルは継続されます。  #warning"慎重に、次の行でエラー"  //しかし、この行はエラーで停止します。  #error"この部分はコンパイルされるべきではありません" |

## サポートされていないその他の機能

C言語からの他のディレクティブ#行および#assert(アサート)、はサポートされていません。

さらに、標準のCプリプロセッサには、\_\_FILE\_\_、\_\_LINE\_\_、\_\_DATE\_\_、\_\_TIME\_\_、および\_\_func\_\_という内部変数があります。これらは、現在の時刻、ソースファイルの名前などのさまざまなパラメータに応じて、コンパイル時に動的にテキスト文字列を作成するために使用されます。このコンパイラには、これらの変数は存在しません。

標準ライブラリ

Cコンパイラは、プログラムが一定の方法で動作する最小限の機能にアクセスできるように、異なる主題(数学、文字列処理など)に対して一連の関数を実装する必要があります。これらの関数は、コードに標準言語ヘッダーを含めることによってプログラムからアクセスできます。

Vircon32の標準Cライブラリには、多数の標準C関数が含まれています。全てではありませんがたとえば、標準Cライブラリにはファイルとディレクタの処理専用の部分があります。ただし、これらのエンティティはこのコンソールには存在しません。

一方、これは特定のコンソール用のコンパイラなのでVircon32標準ライブラリには、オーディオ、ビデオ、ゲームパッドなど、さまざまなコンソールシステムで動作する機能も追加されています。

Vircon32コンパイラの現在のヘッダコレクションは次のとおりです。

## "オーディオ.h"

オーディオチップを操作してサウンドを再生するために使用されます。

## "入力.h"

ゲームパッドの状態を読み取ることができます。

## "math.h(数学.h)"

C言語の最も一般的な数学関数。

## "memcard.h"と入力します。

メモリカードにアクセスして、データを読み取ったり保存したりするために使用します。

## "misc.h(その他のファイル)"

その他の関数:メモリ管理、乱数、ダイナミックメモリ。

## "文字列.h"

テキスト文字列を構築および処理するための関数が含まれています。

## "時間.h"

これにより、時間の流れを測定し、プログラムの速度を制御することができます。

## "ビデオ.h"

ビデオチップにアクセスし、画面上にイメージを表示するために使用されます。

標準Cとのグローバルな違い

## 主な機能

コードの実行は常にこの関数の先頭から開始されるため、main関数は常に存在する必要がある特別な関数です。標準C言語では、main関数のプロトタイプは次のとおりです:

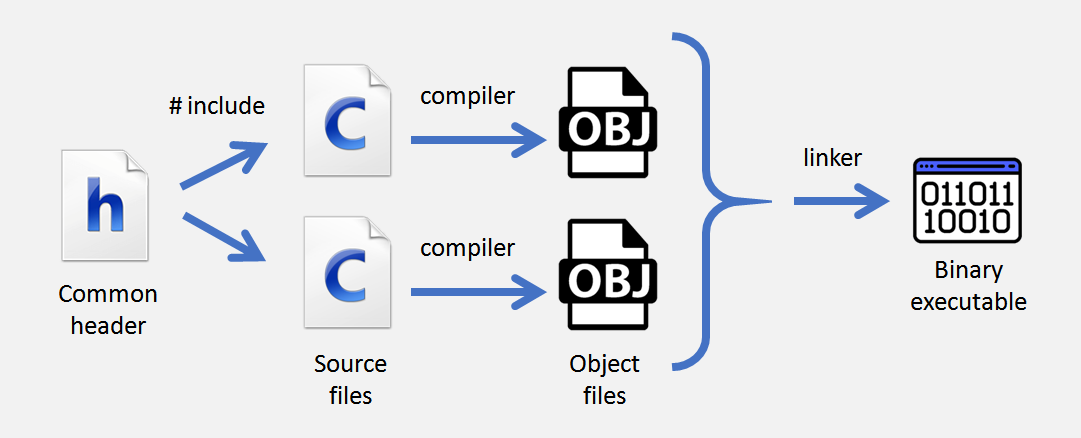
|  |
| --- |
| 内部メイン(内部NumberOfArgumentsイワナ\*引数[])  {を  //.//.//  }をクリックし |

Vircon32は、オペレーティングシステムがなく、最初のBIOSの起動後、マシン上で実行されているのは私たちのプログラムだけであるコンソールです。私たちにパラメータを渡したり、値を返すことができるプログラムは他にないのでVircon32では、そのプロトタイプをmainにしても意味がありません。したがって、このコンパイラのmain関数は、次のように宣言する必要があります。

|  |
| --- |
| 空所関数main()//thisまたは:void main(void)のいずれか  {を  //.//.//  }をクリックし |

## プログラムの構造

標準のCコンパイラでは、プログラムを複数の「コンパイル単位」に分割できます。これは、独自の変数と関数を持つ複数のファイルを個別にコンパイルし、リンカーを使用してリンクできることを意味します。



通常、これらの各ファイルは、それらの関数のプロトタイプを定義する共通ヘッダーを含めることによって、互いの関数にアクセスできますが、それらを実装しません。これは、部分宣言と呼ばれます。

このコンパイラは、より簡略化された構造に従っています。部分宣言がサポートされており、前方宣言に使用できます。しかし、コンパイルされた複数のファイルを結合できるリンカーはありません。ソースファイルが他のファイルの関数を使用できる方法は、メインファイルのそれらのファイルをすべての完全なコードとともに含めることです。

たとえば、標準ライブラリもこの方法で作成され、そのすべての機能の完全な実装が含まれています。

一般的なコンパイラの制限

Vircon32 Cコンパイラは、コンパイラの構築方法に関する限られた知識を持つ1人の人間によってプログラムされています。比較的うまく動作しますが、コンパイルプロセス自体に関しては、より本格的なコンパイラと比較していくつかの制限があります。

## 最適化(Optimization)

このコンパイラは、あまり最適化されたコードを生成しません。最適化するための特定の非常に単純な状況のみを検出します。生成されたアセンブリコードを分析すると、不要な命令を含むセクションを見つけるのは難しくありません。

## エラー検出

コンパイラのエラー検出では、コード内の各エラーの場所が正しく報告されます。ただし、多くの場合、最初に検出されたエラーでプログラムが停止しますが、他のコンパイラでは同時に複数のエラーが発生する場合があります。

## 文字セット

現在、このコンパイラはUnicodeをサポートしていません。一部の英語以外の文字は引き続き使用できますが、ソースファイルはWindows Latin 1/CodePage 1252エンコーディングに従って解釈されます。BIOSに含まれるテキストフォント(標準ライブラリの印刷機能で使用されるフォント)も、この方法でエンコードされます。このエンコードには256文字が含まれ、最初の128文字は標準ASCII文字です。残りは、スペイン語、ポルトガル語、フランス語、ドイツ語のすべての特殊文字と、ユーロなどの一般的に使用される記号です。

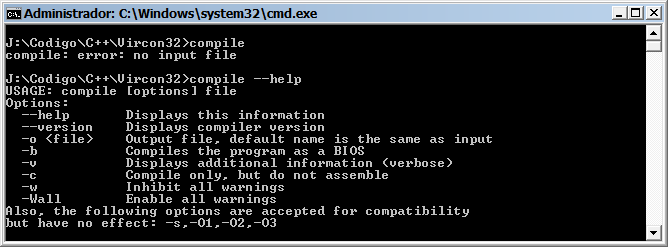
コンパイラの使用

大まかに言って、コンパイラを使用するには2つの方法があり、一般的なVircon32開発ツールのセットがあります。

## コマンドライン

Vircon32ツールはコマンドラインプログラムであるため、プログラムは手動で起動するか、必要に応じてコンパイルするためにスクリプトで呼び出すことができます。コードは任意のテキストエディタで作成できます。コマンドラインを使用するデメリットは、コンパイラがエラーを通知するたびに、メッセージを読み、コード内のエラーを手動で特定する必要があることです。

コンパイラの入力パラメータは、次の図に示すように、gccコンパイラと同じ方法で使用するように選択されています。つまり、コマンド構造は次のようになります。compile.exe[オプション]ファイルです。



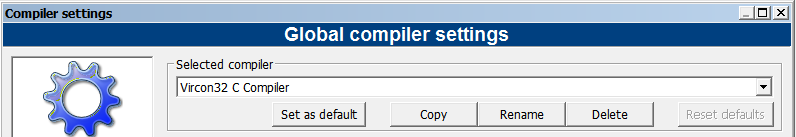
同じパラメータを使用しても、gccと比較した場合、その使用には2つの重要な違いがあります。

* このコンパイラは、アセンブリコード(デフォルトの拡張子は.asm)を生成するだけで、バイナリを作成しません。これを行うには、アセンブラプログラムを呼び出す必要があります。assemble.exe(英語の可能性その後です。
* アセンブラで作成されたバイナリファイルは、Vircon32でも直接実行できません。これは、packerプログラムを使用してバイナリをROMにパッケージ化する必要があるためです。packrom.exeこのようにして、ROMはバイナリとそのイメージおよびサウンド(使用されている場合)を統合します。です。

## IDEでの統合

Vircon32コンパイラのエラーと警告の表示形式も、gccコンパイラと一致するように選択されています。これにより、開発環境が十分に構成可能であれば、コンパイラを開発環境に統合することが容易になります。

たとえば、選択肢の1つは、gccから始まるVircon32コンパイラ用の構成を作成し(gccはどのIDEにも存在するはずです)、それを使用するようにプロジェクトを構成することです。



特に、Code::Blocksの環境では、いくつかの方法でVircon32ツールを操作できます。実際、スクリプトを作成することで、下にカスタムオプションメニュー画像上を追加することもできます。

